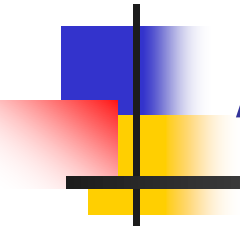


COMP 6231: Distributed System Design

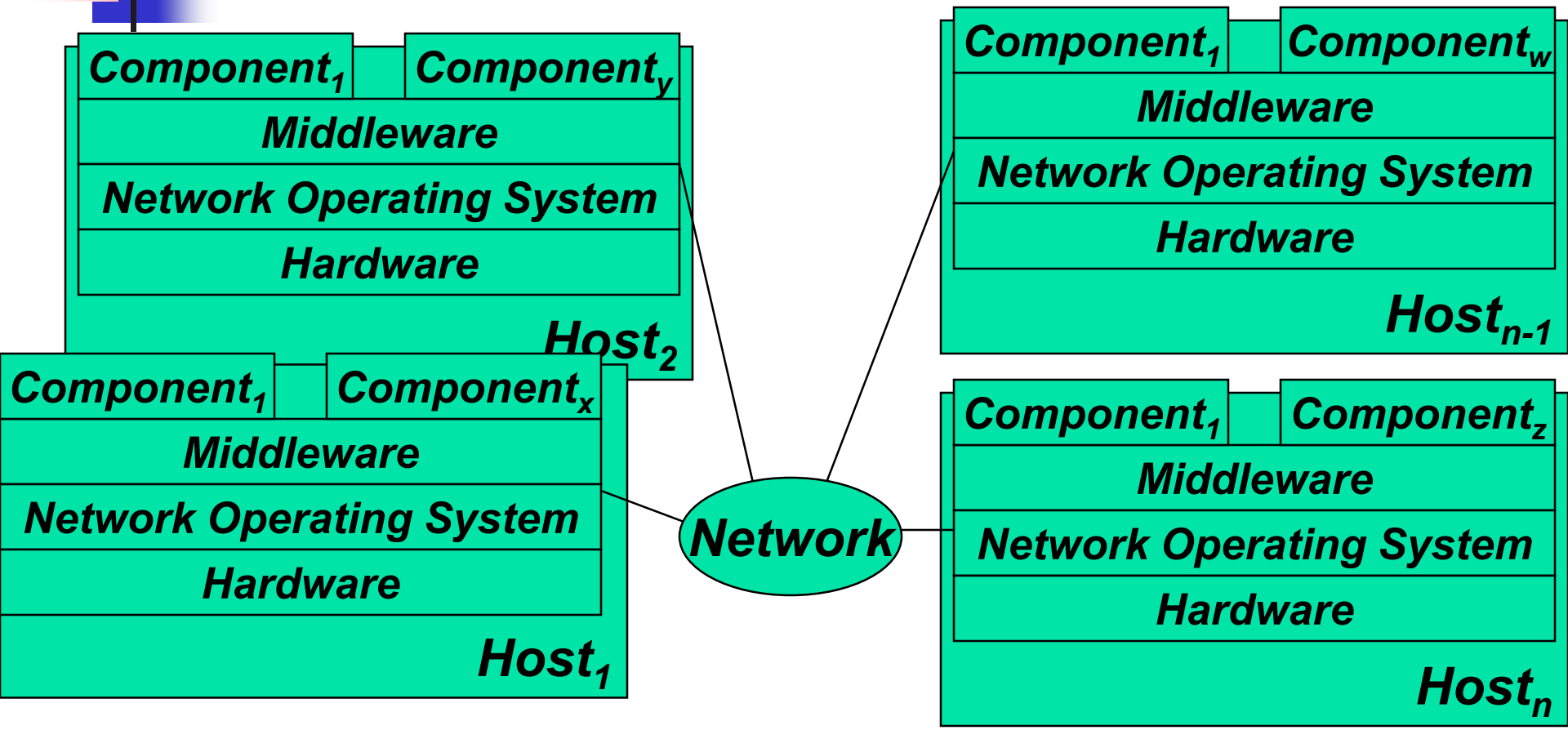
Introduction to Distributed Systems

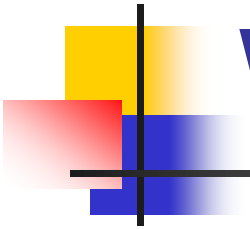
Based on Chapters 1 and 2 of the textbook and slides from

Professor Wolfgang Emmerich, London University

Professor Kenneth P. Birman, Cornell University

What is a Distributed System?

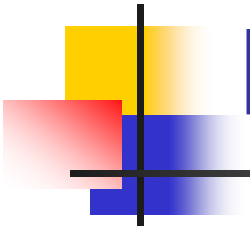




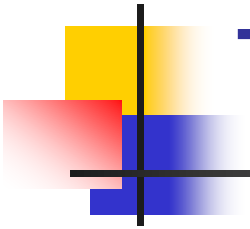
What is a Distributed System?

- A distributed system is a collection of autonomous hosts that that are connected through a computer network. Each host executes components and operates a distribution middleware, which enables the components to coordinate their activities in such a way that *users perceive the system as a single, integrated computing facility.*

How to hide Distribution of Components?



- A components should be able to communicate with any other component the same way.
  - *Access Transparency*
- Components should be able to communicate using their names only without knowing their location
  - *Location Transparency*

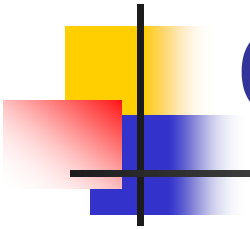


Transparency

- Distributed systems should be perceived by users and application programmers as a whole rather than as a collection of cooperating components.
- Transparency has different dimensions that were identified by ANSA.
- These represent various properties that distributed systems should have.

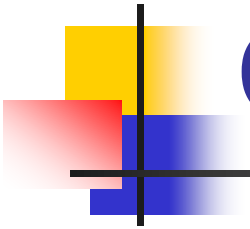
Transparency in a Distributed System

Transparency	Description
Access	Hide differences in data representation and how a resource is accessed
Location	Hide where a resource is located
Migration	Hide that a resource may move to another location
Relocation	Hide that a resource may be moved to another location while in use
Replication	Hide that a resource may be replicated
Concurrency	Hide that a resource may be shared by several competitive users
Failure	Hide the failure and recovery of a resource
Persistence	Hide whether a (software) resource is in memory or on disk



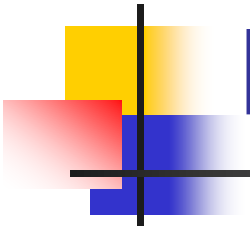
Common Requirements

- Certain requirements are common to many distributed systems
  - Concurrency
  - Resource Sharing
  - Openness
  - Scalability
  - Fault Tolerance
  - Transparency



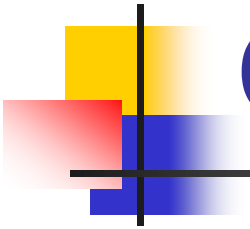
Concurrency

- Components in distributed systems are executed in concurrent processes.
- Components access and update shared resources (e.g. variables, databases, device drivers).
- Integrity of the system may be violated if concurrent updates are not coordinated.
  - Lost updates
  - Inconsistent analysis



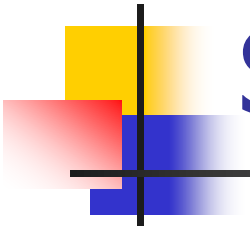
Resource Sharing

- Ability to use any hardware, software or data anywhere in the system.
- Resource manager controls access, provides naming scheme and controls concurrency.
- Resource sharing model (e.g. client/server or object-based) describing how
  - resources are provided,
  - they are used and
  - provider and user interact with each other.



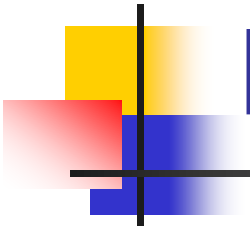
Openness

- Openness is concerned with extensions and improvements of distributed systems.
- Detailed interfaces of components need to be published.
- New components have to be integrated with existing components.
- Differences in data representation of interface types on different processors (of different vendors) have to be resolved.



Scalability

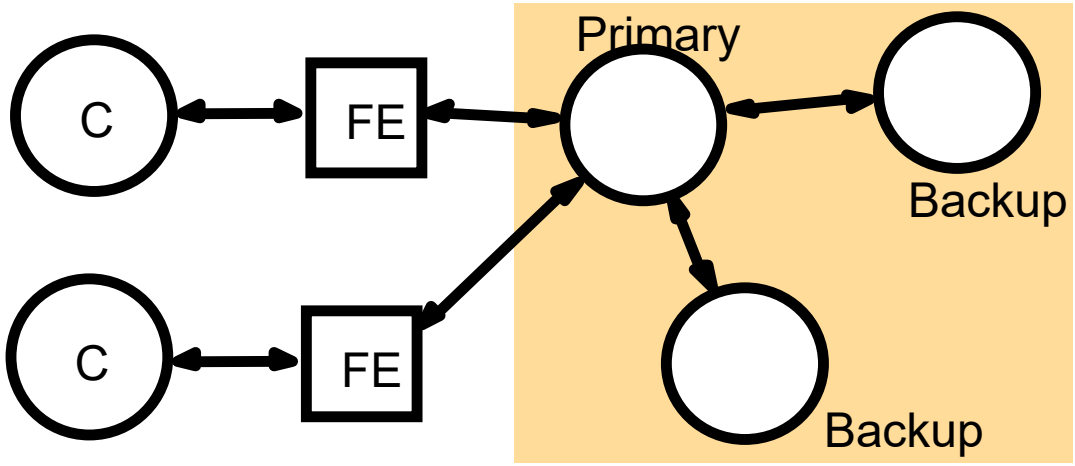
- Adaption of distributed systems to
  - accommodate more users
  - respond faster (this is the hard one)
- Usually done by adding more and/or faster processors.
- Components should not need to be changed when scale of a system increases.
- Design components to be scalable!



Fault Tolerance

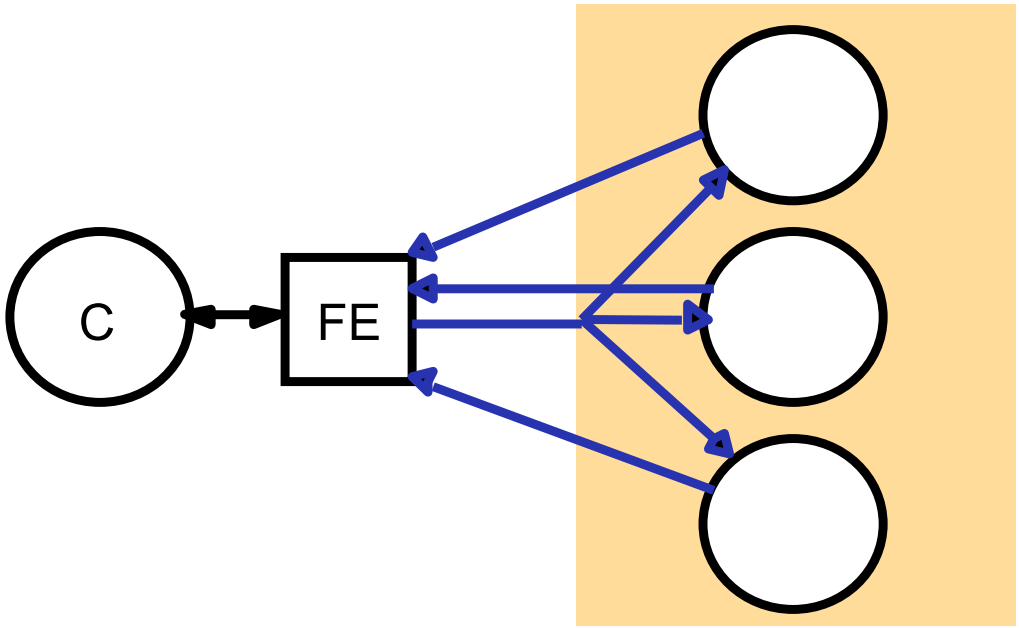
- Hardware, software and networks fail!
- Distributed systems must maintain availability even at low levels of hardware/software/network reliability.
- Fault tolerance is achieved by
  - recovery
  - redundancy

Failure Tolerance



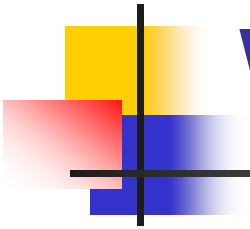
- There is at any time a single primary process and one or more secondary (backup, slave) processes
- FEs communicate with the primary which executes the operation and sends copies of the updated data to the result to backups
- If the primary fails, one of the backups is promoted to act as the primary

High Availability



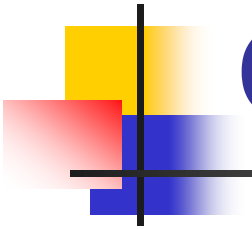
- All the processes perform the same computation
- A single result is produced from the results of all the processes
- Correct result can be produced even when one of the processes produces incorrect result

Why Distributed Systems – What problems are solved?



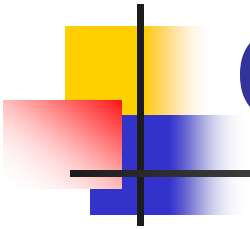
- Easily connect users to remote resources
- Share resources with remote users in a controlled way
  - Hide the fact that the resources are physically distributed over a network -- transparency
  - Should be an open system
    - Offers services by standard rules that describe the syntax and semantics of those services
  - Should be scalable
    - Size, geography, and administration

Distributed System Characteristics



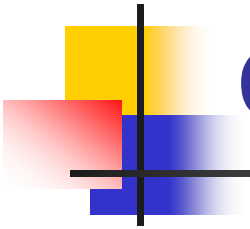
- Multiple autonomous components
- Components are not shared by all users
- Resources may not be accessible
- Software runs in concurrent processes on different processors
- Multiple Points of control
- Multiple Points of failure

Centralized System Characteristics



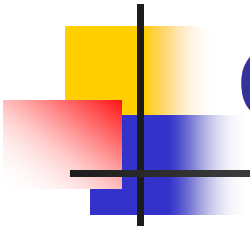
- One component with non-autonomous parts
- Component shared by users all the time
- All resources accessible
- Software runs in a single process
- Single Point of control
- Single Point of failure

Examples of mission-critical distributed applications



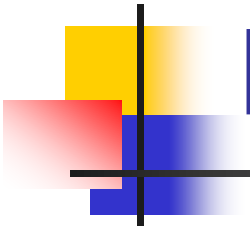
- Banking, stock markets, stock brokerages
- Heath care, hospital automation
- Control of power plants, electric grid
- Telecommunications infrastructure
- Electronic commerce and electronic cash on the Web (very important emerging area)
- Corporate "information" base: a company's memory of decisions, technologies, strategy
- Military command, control, intelligence systems

We depend on distributed systems!



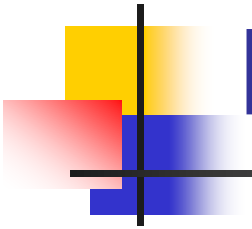
- If these critical systems don't work
  - When we need them
  - Correctly
  - Fast enough
  - Securely and privately
- ... then revenue, health and safety, and national security may be at risk!
- *Should be scalable, failure-tolerant and highly available*

Major Issues in Distributed Systems Design



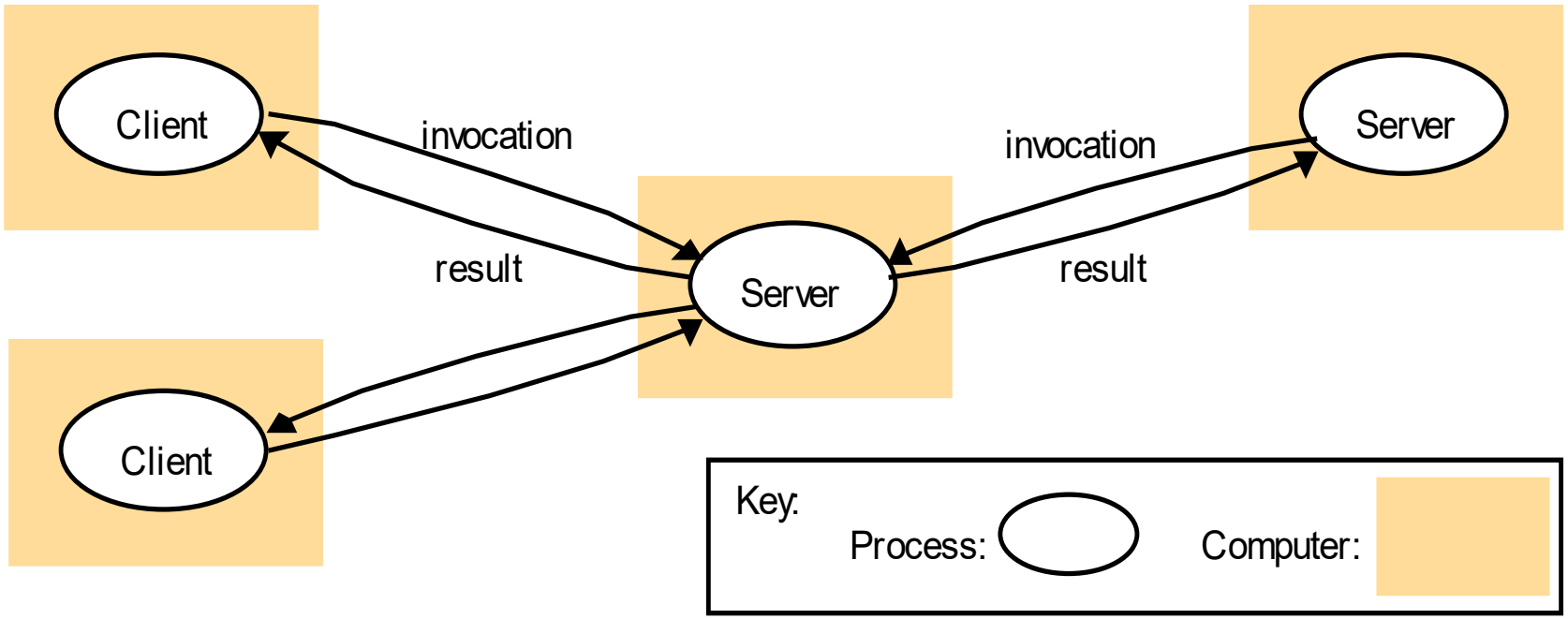
- Independent Failure: Nodes can fail, rest should go on
- Unreliable and Insecure Communication: Connections may fail, messages may be contaminated or modified
- Costly Communication: Communication cost (latency etc) is higher than local communication within a node
- High Availability, Fault-Tolerance are key advantages
- Atomicity is difficult due to absence of Global Time

Distributed Computing Paradigms

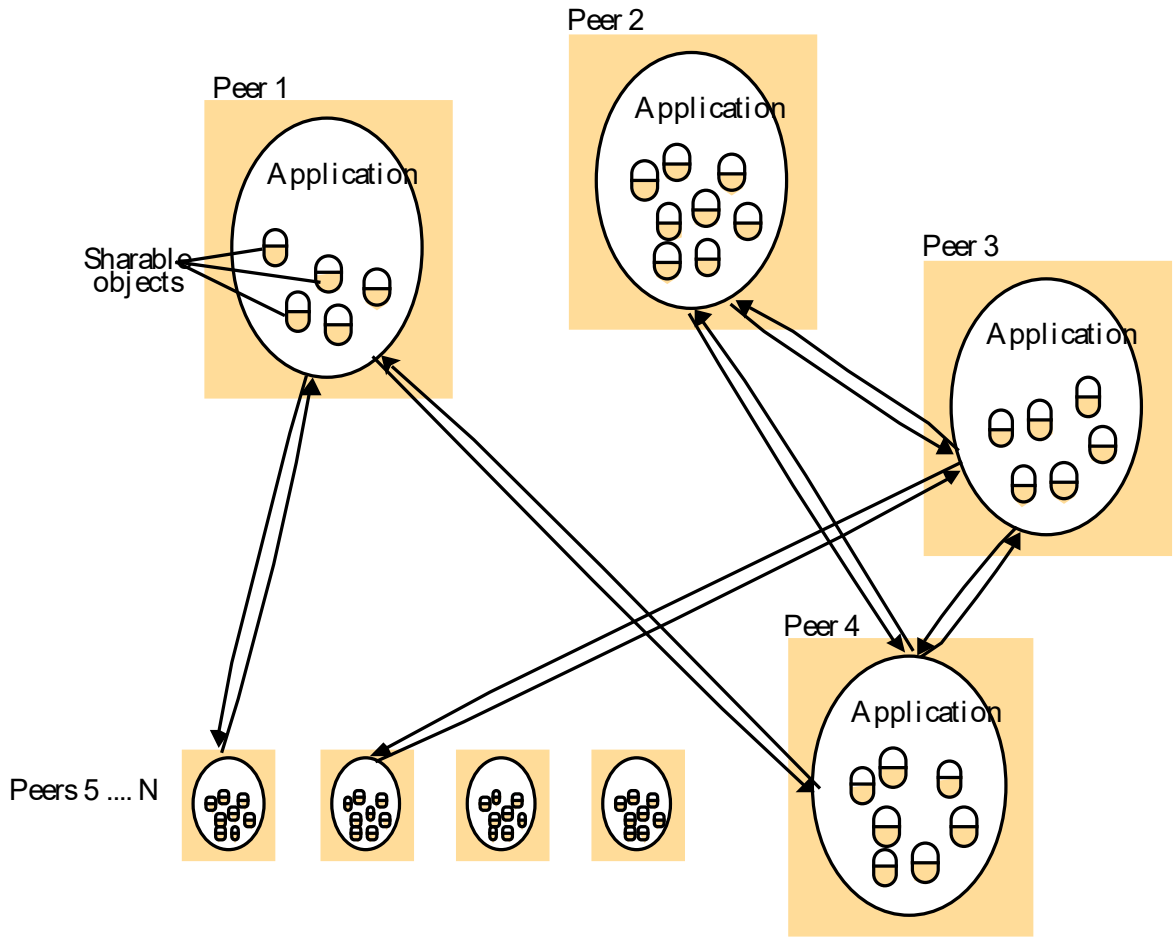


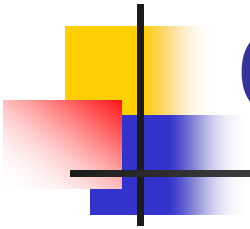
- There are many distributed computing paradigms/architectures
  - Client-Server,
  - Peer-to-Peer,
  - Distributed Agents,
  - Distributed Object Spaces, …
- We will focus on the client-server paradigm only.

Distributed Systems Architecture: Client-Server



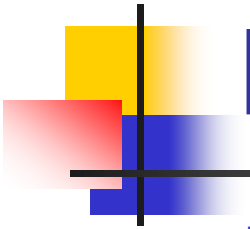
Distributed Systems Architecture: Peer-to-Peer





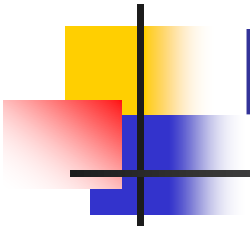
Client-Server Interaction

- Client access to server should be independent of
  - Local or remote server
    - Access Transparency
  - Location of the server
    - Location Transparency
  - Implementation of the server
    - Implementation Transparency,
    - Programming Language Transparency,
    - Replication Transparency, …



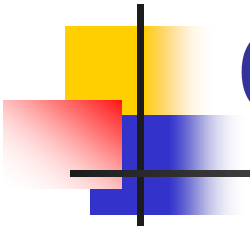
Remote Procedure Call (RPC)

- Pre-RPC: Most applications were built directly over the Internet primitives
  - Their idea: mask distributed computing system using a "transparent" abstraction
    - Looks like normal procedure call
    - Hides all aspects of distributed interaction
    - Supports an easy programming model
- Today, RPC is the core of many distributed systems
- Today, RPC often used from object-oriented systems employing CORBA or COM standards.



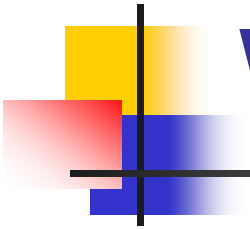
RPC is fundamental

- RMI: Object Oriented RPC
- Network Services: RPC among mostly homogeneous processes (over LANs)
- CORBA: RPC among heterogeneous processes (over WANs using IIOP)
- Web Services: RPC over the web using HTTP



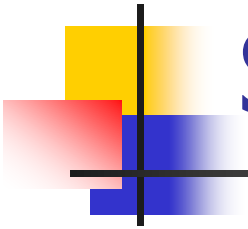
CORBA

- The Common Object Request Broker Architecture (CORBA) is a standard architecture for a distributed objects system.
- CORBA is designed to allow distributed objects to interoperate in a heterogenous environment, where objects can be implemented in different programming language and/or deployed on different platforms



Web Services

- Today, we normally use Web browsers to talk to Web sites
  - Browser names document via URL (lots of fun and games can happen here)
  - Request and reply encoded in HTML, using HTTP to issue request to the site
- Web Services generalize this model so that computers can talk to computers



Server Design

- Many of the server characteristics are achieved through process replication
  - Scalable
    - Automatically create server processes to distribute client requests
  - Fault Tolerant
    - Maintain multiple server processes so that a faulty server can be taken over by a good one
  - Highly Available
    - Maintain multiple server processes so that a client request can be handled without delay